awokado

Release 0.3b17

Aug 13, 2019

User Guide:

1 Installation **3**

2 Quickstart **5**
2.1 Simple example . . . 5

3 Reference **9**

4 Relations **11**

5 Diagram **17**

6 Features **19**
6.1 Filtering . . . 19
6.1.1 syntax . . . 19
6.1.2 available operators . . . 19
6.1.3 examples . . . 19
6.2 Sorting . . . 20
6.2.1 syntax . . . 20
6.2.2 examples . . . 20
6.3 Includes . . . 20
6.3.1 syntax . . . 20
6.3.2 examples . . . 20
6.4 Limit Offset (pagination) . . . 20
6.4.1 syntax . . . 20
6.4.2 examples . . . 20

7 Documentation **21**
7.1 function parameters . . . 21
7.2 examples . . . 21

8 Changelog **23**
8.1 [0.3b16] - 2019-05-24 . . . 23
8.1.1 Changed . . . 23
8.2 [0.3b15] - 2019-05-23 . . . 23
8.2.1 Added . . . 23
8.2.2 Fixes . . . 23
8.2.3 Removed . . . 24

8.3 [0.3b14] - 2019-05-15 . . . 24
8.3.1 Fixes . . . 24
8.4 [0.3b13] - 2019-04-02 . . . 24
8.4.1 Changed . . . 24
8.5 [0.3b12] - 2019-03-14 . . . 24
8.5.1 Added . . . 24
8.5.2 Deprecated . . . 24
8.6 [0.3b11] - 2019-03-13 . . . 25
8.6.1 Added . . . 25
8.7 [0.3b10] - 2019-03-11 . . . 25
8.7.1 Added . . . 25
8.8 [0.3b7] - 2019-03-05 . . . 25
8.8.1 Added . . . 25
8.8.2 Deprecated . . . 25
8.9 [0.3b5] - 2019-03-04 . . . 25
8.9.1 Added . . . 25
8.9.2 Fixes . . . 26
8.10 [0.3b2] - 2019-03-01 . . . 26
8.10.1 Added . . . 26
8.10.2 Changed . . . 26
8.10.3 Deprecated . . . 26
8.10.4 Removed . . . 26

9 Indices and tables **27**

Python Module Index **29**

Index **31**

Fast and flexible low-level API framework based on Falcon , Marshmallow and SQLAlchemy Core API is close to OpenAPI 3.0 specification.

Awokado uses dynaconf for loading it settings. You can find all available variables in *settings.toml* file.

CHAPTER 1

Installation

Use pip at the command line:

```
$ pip install awokado
```

CHAPTER 2

Quickstart

Install awokado before it's too late.

Awokado uses dynaconf for loading it settings. So store them in settings.toml, for example:

```
#settings.toml

[default]
     DATABASE_PASSWORD='your_password'
     DATABASE_HOST='localhost'
     DATABASE_USER='your_user'
     DATABASE_PORT=5432
     DATABASE_DB='try_awokado'
```

2.1 Simple example

Awokado based on Falcon, so we use the REST architectural style. That means we're talking about resources. Resources are simply all the things in your API or application that can be accessed by a URL.

First of all, we need to create a model to further be connected with a resource.

This model will act as a link to a database entity. Read more about it here.

At this point, the database should be already created.

```
#books.py

import falcon
import sqlalchemy as sa
from awokado.consts import CREATE, READ, UPDATE, DELETE
from awokado.db import DATABASE_URL
from awokado.middleware import HttpMiddleware
from awokado.resource import BaseResource
from marshmallow import fields
```

(continues on next page)

(continued from previous page)

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base

Base = declarative_base()


class Book(Base):
    __tablename__ = "books"

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)
    description = sa.Column(sa.Text)
    title = sa.Column(sa.Text)


e = create_engine(DATABASE_URL)
Base.metadata.create_all(e)
```

Resources are represented as classes inherited from awokado BaseResource, that gives an opportunity to use get, create, delete, update methods.

```
class BookResource(BaseResource):
    class Meta:
        model = Book
        name = "book"
        methods = (CREATE, READ, UPDATE, DELETE)

    id = fields.Int(model_field=Book.id)
    title = fields.String(model_field=Book.title, required=True)
    description = fields.String(model_field=Book.description)
```

Add routes, so resources can handle requests:

```
api = falcon.API(middleware=[HttpMiddleware()])

api.add_route("/v1/book/", BookResource())
api.add_route("/v1/book/{resource_id}", BookResource())
```

The final file version should look like this one.

Now we're ready to run the above example. You can use the uwsgi server.

```
pip install uwsgi
uwsgi --http :8000 --wsgi-file books.py --callable api
```

Test it using curl in another terminal.

Create entity using following curl:

```
curl localhost:8000/v1/book --data-binary '{"book":{"title":"some_title","description
↪":"some_description"}}' --compressed -v | python -m json.tool

{
    "book": [
        {
            "description": "some_description",
            "id": 1,
            "title": "some_title"
```

(continues on next page)

(continued from previous page)

```
            }
        ]
    }
```

And then, with read request see what you've got:

```
curl localhost:8000/v1/book | python -m json.tool

 {
     "meta": {
         "total": 1
     },
     "payload": {
         "book": [
             {
                 "description": "some_description",
                 "id": 1,
                 "title": "some_title"
             }
         ]
     }
 }
```

CHAPTER 3

Reference

In class Meta we declare different resource's options. There is a possibility to write your own behavior for certain methods .

class awokado.resource.**BaseResource**

class Meta

Parameters

- **name** – used for two resources connection by relation
- **model** – represents sqlalchemy model or cte
- **methods** – tuple of methods you want to allow
- **auth** – awokado BaseAuth class for embedding authentication logic
- **skip_doc** – set true if you don't need to add the resource to documentation
- **disable_total** – set false, if you don't need to know returning objects amount in read-requests
- **select_from** – provide data source here if your resource use another's model fields (for example sa.outerjoin(FirstModel, SecondModel, FirstModel.id == SecondModel.first_model_id))

auth

alias of *awokado.auth.BaseAuth*

auth(**args*, ***kwargs*) → Tuple[int, str]

this method should return (user_id, token) tuple

create(*session*, *payload: dict*, *user_id: int*) → dict

Create method.

You can override it to add your logic.

First of all, data is prepared for creating: Marshmallow load method for data structure deserialization and then preparing data for SQLAlchemy create a query.

Inserts data to the database (Uses bulky library if there is more than one entity to create). Saves many-to-many relationships.

Returns created resources with the help of read_handler method.

delete (*session*, *user_id: int*, *obj_ids: list*)
Simply deletes objects with passed identifiers.

on_get (*req: falcon.request.Request*, *resp: falcon.response.Response*, *resource_id: int = None*)
Falcon method. GET-request entry point.

Here is a database transaction opening. This is where authentication takes place (if auth class is pointed in resource) Then read_handler method is run. It's responsible for the whole read workflow.

Parameters

- **req** – falcon.request.Request
- **resp** – falcon.response.Response

update (*session*, *payload: dict*, *user_id: int*, **args*, ***kwargs*) → dict
First of all, data is prepared for updating: Marshmallow load method for data structure deserialization and then preparing data for SQLAlchemy update query.

Updates data with bulk_update_mappings sqlalchemy method. Saves many-to-many relationships.

Returns updated resources with the help of read_handler method.

class awokado.auth.**BaseAuth**
CREATE = { 'ROLE NAME HERE': Boolean value }

Example: 'ADMIN': True, 'GUEST': False

READ, UPDATE, DELETE

CHAPTER 4

Relations

Here is a more complicated version of simple awokado usage. Awokado provides you with the possibility to easily build relations between entities.

Let's take the Authors-Books one-to-many relation, for example.

Firstly, we need models:

```
#models.py

import sqlalchemy as sa
from awokado.db import DATABASE_URL
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base


Base = declarative_base()


class Book(BaseModel):
    __tablename__ = "books"

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)
    author_id = sa.Column(
        sa.Integer,
        sa.ForeignKey("authors.id", onupdate="CASCADE", ondelete="SET NULL"),
        index=True,
    )

    description = sa.Column(sa.Text)
    title = sa.Column(sa.Text)

class Author(BaseModel):
    __tablename__ = "authors"

    id = sa.Column(sa.Integer, primary_key=True, autoincrement=True)
```

(continues on next page)

(continued from previous page)

```
    first_name = sa.Column(sa.Text, nullable=False)
    last_name = sa.Column(sa.Text, nullable=False)

e = create_engine(DATABASE_URL)
Base.metadata.create_all(e)
```

Secondly, we write resources for each entity connected with their models.

Bind Book to Author using the ToOne awokado custom_field. Resource argument is the name field of Meta class in Author resource we're connecting to, model_field argument is the field in Book model where Author unique identifier is stored.

```
#resources.py

import sqlalchemy as sa
from awokado import custom_fields
from awokado.consts import CREATE, READ, UPDATE
from awokado.resource import BaseResource
from awokado.utils import ReadContext
from marshmallow import fields

import models as m

class BookResource(BaseResource):
 class Meta:
     model = m.Book
     name = "book"
     methods = (CREATE, READ, UPDATE)

 id = fields.Int(model_field=m.Book.id)
 title = fields.String(model_field=m.Book.title, required=True)
 description = fields.String(model_field=m.Book.description)
 author = custom_fields.ToOne(
     resource="author", model_field=m.Book.author_id
 )
```

The continuation of building the connection is in the Author resource. Here we define another end of connection by the ToMany field.

```
class AuthorResource(Resource):
 class Meta:
     model = m.Author
     name = "author"
     methods = (CREATE, READ, UPDATE)
     select_from = sa.outerjoin(
         m.Author, m.Book, m.Author.id == m.Book.author_id
     )

 id = fields.Int(model_field=m.Author.id)
 books = custom_fields.ToMany(
     fields.Int(),
     resource="book",
     model_field=m.Book.id,
     description="Authors Books",
 )
 books_count = fields.Int(
     dump_only=True, model_field=sa.func.count(m.Book.id)
```

(continues on next page)

(continued from previous page)

```
)
name = fields.String(
    model_field=sa.func.concat(
        m.Author.first_name, " ", m.Author.last_name
    ),
    dump_only=True,
)
last_name = fields.String(
    model_field=m.Author.last_name, required=True, load_only=True
)
first_name = fields.String(
    model_field=m.Author.first_name, required=True, load_only=True
)
```

So finally here are the methods where we add logic for getting connected entities.

```
#BookResource

def get_by_author_ids(
    self, session, ctx: ReadContext, field: sa.Column = None
):
    authors = sa.func.array_remove(
        sa.func.array_agg(m.Author.id), None
    ).label("authors")
    q = (
        sa.select(
            [
                m.Book.id.label("id"),
                m.Book.title.label("title"),
                m.Book.description.label("description"),
                authors,
            ]
        )
        .select_from(
            sa.outerjoin(m.Book, m.Author, m.Author.id == m.Book.author_id)
        )
        .where(m.Book.author_id.in_(ctx.obj_ids))
        .group_by(m.Book.id)
    )
    result = session.execute(q).fetchall()
    serialized_objs = self.dump(result, many=True)
    return serialized_objs


#AuthorResource

def get_by_book_ids(
    self, session, ctx: ReadContext, field: sa.Column = None
):
    books_count = self.fields.get("books_count").metadata["model_field"]
    q = (
        sa.select(
            [
                m.Author.id.label("id"),
                self.fields.get("name")
                .metadata["model_field"]
                .label("name"),
```

(continues on next page)

(continued from previous page)

```
                books_count.label("books_count"),
            ]
        )
        .select_from(
            sa.outerjoin(m.Author, m.Book, m.Author.id == m.Book.author_id)
        )
        .where(m.Book.id.in_(ctx.obj_ids))
        .group_by(m.Author.id)
    )
    result = session.execute(q).fetchall()
    serialized_objs = self.dump(result, many=True)
    return serialized_objs
```

Add routes, so resources can handle requests:

```
app = falcon.API()
api.add_route("/v1/author/", AuthorResource())
api.add_route("/v1/author/{resource_id}", AuthorResource())
api.add_route("/v1/book/", BookResource())
api.add_route("/v1/book/{resource_id}", BookResource())
```

Test it using curl in terminal.

Create entities using following curl:

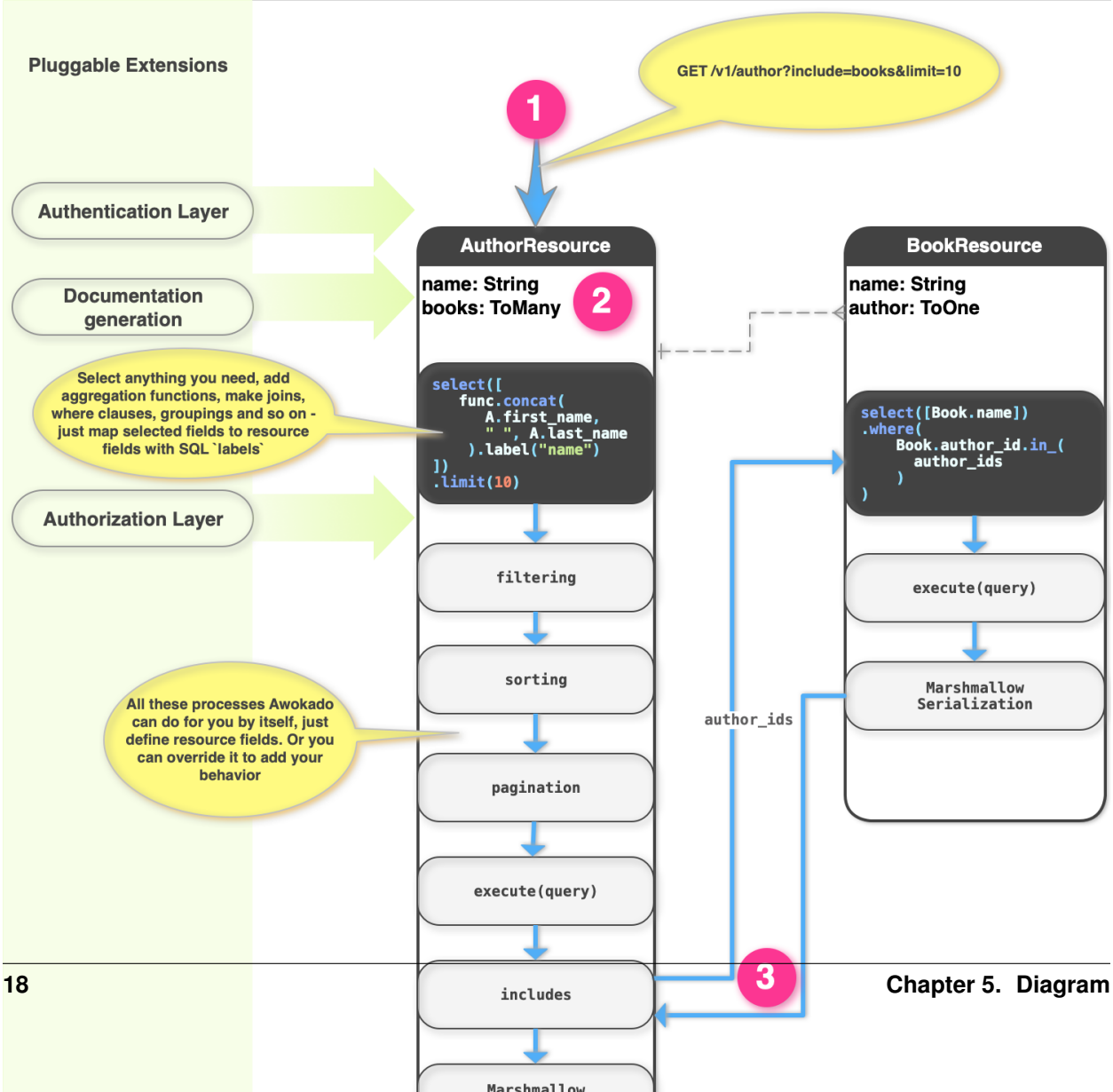
```
  curl localhost:8000/v1/author --data-binary '{"author":{"last_name": "B","first_
↪name": "Sier"}}' --compressed -v | python -m json.tool

  {
      "author": [
          {
              "books": [],
              "books_count": 0,
              "id": 1,
              "name": "Sier B"
          }
      ]
  }

  curl localhost:8000/v1/book --data-binary '{"book":{"title":"some_title",
↪"description":"some_description", "author":"1"}}' --compressed -v | python -m json.
↪tool

  {
      "book": [
          {
              "author": 1,
              "description": "some_description",
              "id": 1,
              "title": "some_title"
          }
      ]
  }
```

And then, with read request see what you've got:

```
curl localhost:8000/v1/author?include=books | python -m json.tool

{
    "meta": {
        "total": 1
    },
    "payload": {
        "author": [
            {
                "books": [
                    1
                ],
                "books_count": 1,
                "id": 1,
                "name": "Sier B"
            }
        ],
        "book": [
            {
                "description": "some_description",
                "id": 1,
                "title": "some_title"
            }
        ]
    }
}

curl localhost:8000/v1/book?include=author | python -m json.tool

{
    "meta": {
        "total": 1
    },
    "payload": {
        "author": [
            {
                "books_count": 1,
                "id": 1,
                "name": "Sier B"
            }
        ],
        "book": [
            {
                "author": 1,
                "description": "some_description",
                "id": 1,
                "title": "some_title"
            }
        ]
    }
}
```

CHAPTER 5

Diagram



CHAPTER 6

Features

6.1 Filtering

6.1.1 syntax

```
resource_field_name[operator]=value
```

6.1.2 available operators

- lte
- eq
- gte
- ilike
- in
- empty
- contains

6.1.3 examples

`/v1/user/?username[ilike]=Andy` it's equal to SQL statement: `SELECT * FROM users WHERE username ILIKE '%Andy%';`

`/v1/user/?id[in]=1,2,3,4` it's equal to SQL statement: `SELECT * FROM users WHERE id IN (1, 2,3,4);`

6.2 Sorting

6.2.1 syntax

sort=`resource_field_name,-another_resource_field_name`

use - for descending order

6.2.2 examples

```
/v1/user/?sort=name,-record_created
```

6.3 Includes

6.3.1 syntax

include=`resource_relation_name`

6.3.2 examples

```
/v1/author/?include=books
/v1/author/?include=books,stores
```

6.4 Limit Offset (pagination)

6.4.1 syntax

limit=`integer`&offset=`integer`

6.4.2 examples

```
/v1/user/?limit=10&offset=10
/v1/user/?offset=10
/v1/user/?limit=2000
```

CHAPTER 7

Documentation

Awokado allows to generate documentation for a project using swagger(3rd version). To generate documentation you need to import generate_documentation function and call it with required parameters. Description of your project can be taken from template, in that case you need to provide path to the template as argument in `template_absolute_path`

7.1 function parameters

- api - your falcon.API instance
- api_host - IP address for your host
- project_name - title for your documentation
- output_dir - path, where swagger doc will be added
- api_version `default "1.0.0"` - string with number of version of you project
- template_absolute_path `default None` - absolute path to template with description of your project

7.2 examples

```
from awokado.documentation import generate_documentation
from dynaconf import settings
from api.routes import api

generate_documentation(
    api=api,
    api_host=settings.MY_HOST_FOR_DOCUMENTATION,
    api_version="2.0.0",
    project_name="API Documentation",
    template_absolute_path="Users/my_user/projects/my_project/template.tmpl",
```

(continues on next page)

(continued from previous page)

```
    output_dir="my_project/documentation",
)
```

CHAPTER 8

Changelog

The format is based on Keep a Changelog

8.1 [0.3b16] - 2019-05-24

8.1.1 Changed

Updated falcon from 1.4.1 to 2.0.0 (Falcon 2 Changelog), which led to following changes:

- `application.req_options.auto_parse_qs_csv` param is now `False` by default, so you'll need to manually set it to `True`
- `application.req_options.strip_url_path_trailing_slash` param is now `False` by default, so you'll need to manually set it to `True`
- For direct data read from request `req.bounded_stream` is now used instread of `req.stream`.

8.2 [0.3b15] - 2019-05-23

8.2.1 Added

- Added pre-commit hook to run black formatting checks
- Added ability to specify full database url in settings

8.2.2 Fixes

- Fixed "load_only" fields appearing in read request results
- Fixed "awokado_debug" setting being always required in settings

- Fixed attribute "auth" being mandatory in resource.Meta
- Fixed method "auth" being mandatory to overwrite in resource
- Fixed method "audit_log" being mandatory to overwrite in resource

8.2.3 Removed

- Functions `set_bearer_header`, `get_bearer_payload` and `AWOKADO_AUTH_BEARER_SECRET` var are removed

8.3 [0.3b14] - 2019-05-15

8.3.1 Fixes

- Fixed documentation generation for bulk operations

8.4 [0.3b13] - 2019-04-02

8.4.1 Changed

No backward compatibility. Need to change custom delete() and can_create() methods.

- `delete` method supports bulk delete.
- add `payload` attribute to `can_create` method.

8.5 [0.3b12] - 2019-03-14

8.5.1 Added

- `select_from` attribute in class Meta, allows you to specify `sqlalchemy.select_from()` arguments. Example:

```
class AuthorResource(Resource):
    class Meta:
        model = m.Author
        name = "author"
        methods = (CREATE, READ, UPDATE, BULK_UPDATE, DELETE)
        auth = None
        select_from = sa.outerjoin(
            m.Author, m.Book, m.Author.id == m.Book.author_id
        )
```

8.5.2 Deprecated

- `join` argument in the resource field

8.6 [0.3b11] - 2019-03-13

8.6.1 Added

- ability to make list of joins in resource field

```
    book_titles = fields.List(
    fields.Str(),
    resource="author",
    model_field=sa.func.array_remove(sa.func.array_agg(m.Book.title), None),
    join=[
        OuterJoin(
            m.Tag, m.M2M_Book_Tag, m.Tag.id == m.M2M_Book_Tag.c.tag_id
        ),
        OuterJoin(
            m.M2M_Book_Tag, m.Book, m.M2M_Book_Tag.c.book_id == m.Book.id
        ),
    ],
)
```

8.7 [0.3b10] - 2019-03-11

8.7.1 Added

- Automated SQL generation for `POST/PATCH` requests

8.8 [0.3b7] - 2019-03-05

8.8.1 Added

- `bulk_create` method in base resource

8.8.2 Deprecated

- `create` method (is going to be replaced with bulk_create)

8.9 [0.3b5] - 2019-03-04

8.9.1 Added

- API simple workflow diagram
- `disable_total` attr for Resource.Meta. Set it to `True` to avoid adding total column: `sa.func.count().over()`. Useful for `historical` tables, where pagination based on date instead of limit / offset to not overload SQL database

8.9.2 Fixes

- Fixed `description` arg for `ToMany` and `ToOne` fields (was broken)

8.10 [0.3b2] - 2019-03-01

8.10.1 Added

- Documentation generation for API resources
- Automated SQL generation for `GET` requests (including ToOne and ToMany relation fields)
- AWOKADO_DEBUG handle traceback exception in API response

8.10.2 Changed

- all `Forbidden` exceptions now raise HTTP_403 instead of HTTP_401

8.10.3 Deprecated

-

8.10.4 Removed

- ### Fixed
-

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

a

awokado.auth, 10

Index

A

auth (*awokado.resource.BaseResource.Meta attribute*), 9
auth() (*awokado.resource.BaseResource method*), 9
awokado.auth (*module*), 10

B

BaseAuth (*class in awokado.auth*), 10
BaseResource (*class in awokado.resource*), 9
BaseResource.Meta (*class in awokado.resource*), 9

C

create() (*awokado.resource.BaseResource method*), 9

D

delete() (*awokado.resource.BaseResource method*), 10

O

on_get() (*awokado.resource.BaseResource method*), 10

U

update() (*awokado.resource.BaseResource method*), 10